Chapter I : Familiarization with the basics of Python programming

🔶 1. Introduction to Python

- **Python** is a **high-level, interpreted programming language**.
- Created by **Guido van Rossum** and released in **1991**.
- Used for web development, data science, AI, automation, etc.

🔶 2. Features of Python

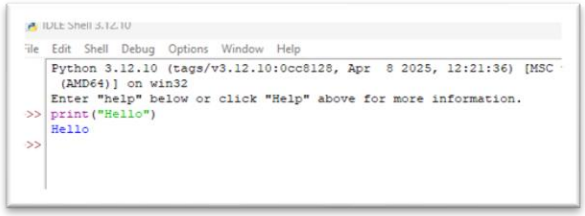
- **Easy to learn & read**
- **Interpreted**: No need to compile
- **Object-Oriented**
- **Free and Open Source**
- **Portable**: Run on any platform
- **Extensive Library Support**

🔶 3. Execution Modes

1. Interactive Mode

- In this mode, you type Python commands **line by line**, and they are **executed immediately**.
- It is useful for **quick testing and learning**.

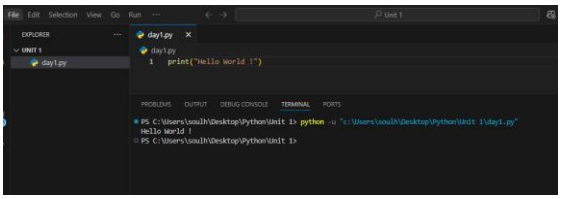
Example:


Edit Shell Debug Options Window Help
Python 3.12.10 (tags/v3.12.10:0cc8128, Apr 8 2025, 12:21:36) [MSC
(AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>> print("Hello")
Hello
>>

2. Script Mode

- In this mode, you **write the entire program in a .py file** (script) and run it all at once.
- Best for writing **longer programs**.

Example :



🔶 4. Python Tokens

Tokens are the **smallest units** in a Python program.

☑ Types of Tokens:

Token Type	Description	Examples
Keywords	Reserved words with special meaning	if, else, while, def
Identifiers	Names for variables/functions	num, total, printData
Literals	Constant values	10, 'hello', 3.14, True
Operators	Symbols for operations	+, -, *, /, ==
Punctuators	Special symbols for structure	:, (), [], {}, ,

🔶 5. Variables

- A variable serves as a named container or label that points to a specific piece of data stored in the computer's memory. Instead of directly interacting with memory addresses, which are complex and difficult for humans to manage, variables provide a convenient and readable way to store, reference, and manipulate data within a program.

```
x = 5
name = "John"
```

☑ Rules:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)
- A variable name cannot be any of the Python keywords.

🔶 6. L-value and R-value

Term	Meaning	Example
l-value	Variable name (left side of =)	x = 10 → x
r-value	Actual value (right side of =)	x = 10 → 10

🔶 7. Comments in Python

- Comments are used to **explain code**, not executed.

Single-line Comments:

```
# This is a single-line comment
```

Multi-line Comments:

```
'''
This is a
multi-line comment
'''
```

Chapter II: Knowledge of data types

🔶 What is a Data Type?

Data types are the classification or categorization of data items. It represents the kind of value that tells what operations can be performed on a particular data. Since everything is an object in Python programming, Python data types are classes and variables are instances (objects) of these classes.

1. Number Data Types

These store **numeric values**.

🔹 a. Integer (int)

- Whole numbers, no decimals.
- Examples:

```
x = 10
y = -5
```

🔹 b. Floating Point (float)

- Numbers with decimal points.
- Examples:

```
pi = 3.14
temp = -7.5
```

🔹 c. Complex (complex)

- Numbers with a real and imaginary part.
- Examples:

```
z = 2 + 3j
```

- z.real → 2.0, z.imag → 3.0

---

2. Boolean (bool)

Represents **True or False** values.

```
a = True
b = False
```

- Often used in **conditions** and **comparisons**.

☑ 3. Sequence Data Types

These store **multiple values in an ordered manner**.

🔹 a. String (str)

- Collection of characters (text).
- Enclosed in ' or ".

```
name = "Chandramouli"
age = 21
```

🔹 b. List (list)

- Ordered, **mutable** (changeable).
- Can hold different data types.

```
fruits = ["apple", "banana", "cherry"]
fruits[0] = "orange"
```

🔹 c. Tuple (tuple)

- Ordered, **immutable** (unchangeable).

```
colors = ("red", "green", "blue")
# colors[0] = "yellow" → Error
```

☑ 4. None Type

- Represents **no value** or **null**.

```
x = None
```

- Used to show absence of value.

☑ 5. Mapping Type – Dictionary (dict)

- Stores data as **key-value pairs**.

```
student = {
    "name": "Amit",
    "age": 17,
    "class": 12
}
print(student["name"])  # Output: Amit
```

☑ 6. Mutable vs Immutable Data Types

Type	Can be Changed?	Example
Mutable	☑ Yes	list, dict, set
Immutable	❌ No	int, float, str, tuple

🔹 **Mutable Example:**

```
marks = [90, 80]
marks[1] = 85
```

🔹 **Immutable Example:**

```
name = "Rahul"
# name[0] = "K" → Error
```

🧠 **Quick Summary:**

Category	Types
Number	int, float, complex
Boolean	bool
Sequence	str, list, tuple
Mapping	dict
Special	NoneType (None)
Mutability	Mutable: list, dict; Immutable: str, tuple, int

Practice :

1) What is the data type of: `5 + 3.0`?
2) Which data type would you use to store: `"Hello"`?
3) Name any **mutable** data type.
4) Is `tuple` mutable or immutable?
5) What does the `None` keyword represent in Python?

i. Write a program to print your **name**, **age**, and **school name**.

ii. Create a list of your **favourite subjects** and print the list.

iii. Create a tuple of any **5 animals** and print each animal on a separate line.

iv. Create a dictionary to store your:

   a. Name

   b. Age

   c. Class

   d. Marks in 3 subjects and Print all the values.

v. Create a list of 5 fruits and print the **first** and **last** fruits using indexing.

vi. Create a list of 5 hobbies. Replace the 3rd hobby with a new one and print the updated list.

Chapter III: Operators

What are Operators?

Operators in Python are special symbols or keywords that perform operations on one or more values or variables, known as operands. They are fundamental building blocks for creating expressions and controlling program flow.

- **1. Arithmetic Operators**

Used for basic **mathematical calculations**.

Operator	Meaning	Example	Output
+	Addition	5 + 3	8
-	Subtraction	10 - 4	6
*	Multiplication	2 * 5	10
/	Division (float)	10 / 2	5.0
//	Floor Division	10 // 3	3
%	Modulus (remainder)	10 % 3	1
**	Exponent (power)	2 ** 3	8

- **2. Relational (Comparison) Operators**

Used to **compare** two values. Result is **True or False**.

Operator	Meaning	Example	Output
==	Equal to	5 == 5	True
!=	Not equal to	5 != 3	True
>	Greater than	7 > 3	True
<	Less than	2 < 1	False
>=	Greater or equal	4 >= 4	True
<=	Less or equal	6 <= 5	False

- **3. Logical Operators**

Used to combine **multiple conditions**.

Operator	Meaning	Example	Output
and	True if both are True	5 > 3 and 4 < 6	True
or	True if at least one is True	4 > 5 or 7 > 6	True
not	Reverses the result	not(5 > 3)	False

- **4. Assignment Operators**

Used to **assign values** to variables.

Operator	Meaning	Example
=	Assign value	x = 10

🔹 5. Augmented Assignment Operators

Shortcuts for performing operations and assignment together.

Operator	Meaning	Example	Same As
+=	Add and assign	x += 2	x = x + 2
-=	Subtract and assign	x -= 3	x = x - 3
*=	Multiply and assign	x *= 2	x = x * 2
/=	Divide and assign	x /= 4	x = x / 4
//=	Floor divide and assign	x //= 2	x = x // 2
%=	Modulus and assign	x %= 3	x = x % 3
**=	Exponent and assign	x **= 2	x = x ** 2

🔹 6. Identity Operators

Used to check if **two variables refer to the same object** in memory.

Operator	Meaning	Example	Output
is	True if same object	a is b	True/False
is not	True if not same object	a is not b	True/False

📍 Example:

```
x = [1, 2]
y = x
z = [1, 2]
print(x is y)     # True (same object)
print(x is z)     # False (different object, same value)
```

🔹 7. Membership Operators

Used to check if a **value exists in a sequence** (like list, string, tuple).

Operator	Meaning	Example	Output
in	True if value is present	'a' in 'apple'	True
not in	True if value is not present	'x' not in 'ball'	True

Type of Operator	Examples
Arithmetic	+, –, *, /, **, %
Relational	==, !=, <, >, <=
Logical	and, or, not
Assignment	=, +=, –=, *= etc.
Identity	is, is not
Membership	in, not in

Operators	Associativity
() Highest precedence	Left - Right
**	Right - Left
+x , -x, ~x	Left - Right
*, /, //, %	Left - Right
+, -	Left - Right
<<, >>	Left - Right
&	Left - Right
^	Left - Right
|	Left - Right
Is, is not, in, not in, <, <=, >, >=, ==, !=	Left - Right
Not x	Left - Right
And	Left - Right
Or	Left - Right
If else	Left - Right
Lambda	Left - Right
=, +=, -=, *=, /= Lowest Precedence	Right - Left

Practice Questions:

🔢 Arithmetic Operators

1. Add two numbers and print the result.
2. Subtract 15 from 45 and print the result.
3. Multiply 7 and 8.
4. Divide 20 by 4 and show the result.
5. Find the remainder when 17 is divided by 3.
6. Calculate 2 raised to the power 5.

🔁 Assignment Operators

7. Assign value 10 to a variable and increase it by 5 using +=.
8. Take a number, multiply it by 3 using *= and print the result.

🔍 Relational Operators

9. Check if 10 is greater than 7.
10. Check if 12 is equal to 15.

🧠 Logical Operators

11. Use and to check if 5 > 3 and 8 < 10.
12. Use or to check if 4 < 2 or 6 > 1.
13. Use not to reverse the result of 10 > 3.

💉 Identity Operators

14. Create two variables a = 5, b = 5, check if a is b.
15. Create x = [1, 2], y = [1, 2], check if x is y.

🔎 Membership Operators

16. Check if "a" is in "apple".
17. Check if 3 is in the list [1, 2, 3, 4].
18. Check if "cat" is **not** in the list ["dog", "cow", "goat"].

Chapter IV: Expressions, statement, type conversion, and input/output

1. Expression

- An **expression** is a combination of **variables**, **operators**, and **values** that results in a **single value**.

Example:

```
x = 5 + 2 * 3
# 2 * 3 = 6 → 5 + 6 = 11
```

2. Statement

- A **statement** is a complete instruction that performs an action.

Examples:

```
x = 10              # Assignment statement
print(x)            # Output statement
if x > 5:           # Conditional statement
    print("Yes")
```

3. Precedence of Operators

- Determines the **order in which operations** are performed in an expression.

Operators	Associativity
() Highest precedence	Left - Right
**	Right - Left
+x , -x, ~x	Left - Right
*, /, //, %	Left - Right
+, -	Left - Right
<<, >>	Left - Right
&	Left - Right
^	Left - Right
|	Left - Right
Is, is not, in, not in, <, <=, >, >=, ==, !=	Left - Right
Not x	Left - Right
And	Left - Right
Or	Left - Right
If else	Left - Right
Lambda	Left - Right
=, +=, -=, *=, /= Lowest Precedence	Right - Left

4. Evaluation of Expression

- Python follows **operator precedence** and **left-to-right associativity** to evaluate expressions.

Example:

```
result = 2 + 3 * 4
# 3 * 4 = 12 → 2 + 12 = 14
```

🔶 5. Type Conversion

Changing data from one type to another.

🔹 a. Implicit Conversion (Automatic)

- Python converts **smaller type to larger type** automatically.

```
x = 5       # int
y = 2.5     # float
z = x + y   # z becomes float (7.5)
```

🔹 b. Explicit Conversion (Type Casting)

- Done **manually using functions** like int(), float(), str().

```
a = "10"
b = int(a)   # string to integer
```

🔶 6. Input and Output in Python

🔹 Input from User

- Use input() function to take user input.

```
name = input("Enter your name: ")
num = int(input("Enter an integer: "))
num = str(input("Enter an string: "))
```

Note: input() always returns **string**, even if you enter a number.

🔸 Convert input to int or float:

```
age = int(input("Enter your age: "))
marks = float(input("Enter your marks: "))
```

🔹 Output to Console

- Use print() function to display output.

```
print("Hello!")
print("Your name is", name)
```

☑ Quick Summary Table

Concept	Example
Expression	3 + 4 * 2
Statement	x = 5, print(x)
Implicit Conversion	int + float → float
Explicit Conversion	int("10"), float("5.5")
Input	input("Enter name: ")
Output	print("Hello!")

Practice Q.

1) Write a Python **expression** to calculate the area of a rectangle (length × breadth).

2) Identify whether the following is an **expression** or a **statement**:

```
total = marks1 + marks2
```

3) What will be the output of the following code?

```
x = 2 + 3 * 4
print(x)
```

- **2. Operator Precedence**

4. Evaluate the following expression step by step and write the result:

```
result = 5 + 2 * 3 ** 2
```

5. Write a Python program to show that parentheses change the result:

   Without parentheses: 5 + 3 * 2

   With parentheses: (5 + 3) * 2

- **3. Type Conversion**

6. What is the data type of the result of this code?

```
a = 5
b = 2.0
c = a + b
print(type(c))
```

7. Convert the string "100" to an integer and multiply it by 2.

8. Write a program that takes a number as input using input() and converts it into float using float().

- **4. Input and Output**

9. Write a program that:
   - Takes your name as input
   - Takes your age as input (as an integer)
   - Prints a greeting message like:
     "Hello [name], you are [age] years old!"

10. Write a program to take two numbers as input from the user and print their sum, difference, and product.

Chapter V: Errors

What is an Error?

An **error** is a problem in a program that causes it to **stop running** or **give incorrect output**.

There are mainly **three types of errors** in Python:

🔴 1. Syntax Errors

📌 **Definition:**

Syntax error occurs when the code doesn't follow Python's rules, like using incorrect grammar in English. Python stops and points out the issue before running the program.

```
💡 Example:
print("Hello"
# Missing closing parenthesis → SyntaxError

🔧 Output:
SyntaxError: unexpected EOF while parsing
```

🟠 2. Logical Errors

📌 **Definition:**

Errors that occur when the **program runs without crashing**, but the **output is incorrect** due to a mistake in logic.

```
💡 Example:
# Finding average, but forgot to divide by 3
x = 10
y = 20
z = 30
avg = x + y + z      # Logical error
print("Average =", avg)

🔧 Output:
Average = 60   # Wrong! It should be 20
```

🟡 3. Run-time Errors (Exceptions)

📌 **Definition:**

Errors that occur **while the program is running**, usually due to **invalid input or operations**.

```
💡 Example:
a = int(input("Enter a number: "))
b = 0
print(a / b)
🔧 Output:
ZeroDivisionError: division by zero
```

☑ **How to fix:**

- Use **try-except** blocks to handle such errors safely:

```
try:
    a = int(input("Enter a number: "))
    b = int(input("Enter another number: "))
    print(a / b)
except ZeroDivisionError:
    print("You can't divide by zero!")
```

☑ **Quick Summary Table**

Error Type	When it Occurs	Causes	Example Problem
Syntax Error	While writing the code	Wrong grammar/syntax	Missing) or :
Logical Error	During execution (correct syntax)	Wrong logic/formula	Wrong average calculation
Run-time Error	While the program is running	Invalid operations	Divide by zero, wrong input

Practice Q.

🔹 **Question 1**

print("Hello, world!"
What type of error is this?
Your Answer: _____

🔹 **Question 2**

x = 10
y = 0
print(x / y)
What type of error is this?
Your Answer: _____

🔹 **Question 3**

a = 5
b = 3
average = a + b / 2
print("Average:", average)
What type of error is this?
Your Answer: _____

- **Question 4**

```
name = input("Enter your name: ")
print("Welcome, " + name)
```

What type of error is this?
Your Answer: _____

- **Question 5**

```
if x > 10
print("x is big")
```

What type of error is this?
Your Answer: _____

- **Question 6**

```
a = int("abc")
```

What type of error is this?
Your Answer: _____

- **Question 7**

```
radius = 5
area = 3.14 * radius * height
print("Area:", area)
```

What type of error is this?
Your Answer: _____

- **Question 8**

```
x = 7
y = 3
z = x + y
print(z)
```

What type of error is this?
Your Answer: _____

- **Question 9**

```
print("Today's date is: " + 20)
```

What type of error is this?
Your Answer: _____

- **Question 10**

```
marks = [90, 85, 78]
print(marks(1))
```

What type of error is this?
Your Answer: _____

Chapter VI: Conditional statements

☑ What are Conditional Statements?

Conditional statements in Python are programming constructs that allow for the execution of specific blocks of code based on whether certain conditions are met. They are fundamental for controlling the flow of a program, enabling it to make decisions and behave dynamically in response to different inputs or states. They use **if, if-else, and if-elif-else** structures.

🔹 1. if Statement

Executes a block of code **only if** the condition is true.

```
📍 Syntax:
if condition:
    # code to run

📍 Example:
age = 18
if age >= 18:
    print("You are eligible to vote.")
```

🔹 2. if-else Statement

Executes one block of code **if condition is true**, another block **if false**.

```
📍 Syntax:
if condition:
    # code if true
else:
    # code if false

📍 Example:
num = 5
if num % 2 == 0:
    print("Even")
else:
    print("Odd")
```

🔹 3. if-elif-else Statement

Used for **multiple conditions**.

```
📍 Syntax:
if condition1:
    # code
elif condition2:
    # code
```

```
else:
    # code
📍 Example:
marks = 85
if marks >= 90:
    print("Grade A")
elif marks >= 75:
    print("Grade B")
else:
    print("Grade C")
```

🔁 **Flowchart for if-else:**

```
     Start
       |
   [Condition]
   /        \
True       False
 |           |
[Block1]   [Block2]
 |           |
  \         /
      End
```

🧩 **Simple Programs**

☑️ **Program 1: Absolute Value**

```
num = int(input("Enter a number: "))
if num < 0:
    num = -num
print("Absolute value is:", num)
```

☑️ **Program 2: Sort 3 Numbers**

```
a = int(input("Enter first number: "))
b = int(input("Enter second number: "))
c = int(input("Enter third number: "))

if a <= b and a <= c:
    if b <= c:
        print(a, b, c)
    else:
        print(a, c, b)
elif b <= a and b <= c:
```

```
    if a <= c:
        print(b, a, c)
    else:
        print(b, c, a)
else:
    if a <= b:
        print(c, a, b)
    else:
        print(c, b, a)
```

☑ **Program 3: Check Divisibility**

```
num = int(input("Enter a number: "))
if num % 5 == 0 and num % 3 == 0:
    print("Divisible by both 3 and 5")
elif num % 5 == 0:
    print("Divisible by 5 only")
elif num % 3 == 0:
    print("Divisible by 3 only")
else:
    print("Not divisible by 3 or 5")
```

☑ **Summary Table**

Statement	Use Case
if	One condition
if-else	One condition, two outcomes
if-elif-else	Multiple conditions (more than 2 cases)

Practice Q.

🔹 Question 1

Write a program to check whether a number is **positive or negative**.

🔹 Question 2

Take a number from the user and check whether it is **even or odd**.

🔹 Question 3

Write a program to check if a number is **divisible by 3 and 5**.

🔹 Question 4

Write a program to take **two numbers** and print the **greater one**.

🔹 Question 5

Write a program to take a number and print:

- "Positive" if number > 0
- "Zero" if number == 0
- "Negative" if number < 0

🔹 Question 6

Write a program to check if a person is **eligible to vote** (age ≥ 18).

🔹 Question 7

Take 3 numbers from the user and print them in **ascending order**.

🔹 Question 8

Write a program to find the **absolute value** of a number.

🔹 Question 9

Write a program to input a mark and print:

- "Pass" if mark ≥ 33
- "Fail" otherwise

🔹 Question 10

Write a program to assign a **grade** based on marks:

- 90 and above – Grade A
- 75 to 89 – Grade B
- 60 to 74 – Grade C
- Below 60 – Grade D

Chapter VII: Iterative Statement

🔹 1. for Loop

A for loop in Python is used for iterating over a sequence (such as a list, tuple, string, or dictionary) or other iterable objects. It allows you to execute a block of code once for each item in the sequence.

```
Syntax:
for variable in sequence:
    # code block

Example:
for i in range(5):
    print(i)

Output:
0
1
2
3
4
```

🔹 2. range() Function

Used to generate a sequence of numbers.

```
Syntax:
range(start, stop, step)

Examples:
range(5)
# → 0 to 4
range(1, 6)
# → 1 to 5
range(2, 10, 2)
# → 2, 4, 6, 8
```

🔹 3. while Loop

Repeats a block of code **while a condition is true**.

```
🔸 Syntax:

while condition:
    # code block
```

```
Example:

i = 1
while i <= 5:
    print(i)
    i += 1
```

- **4. Flowchart of a while loop:**

```
     Start
       |
 [Check Condition]
     |
   /   \
True   False
  |      |
[Block]   End
 |
Go back to condition
```

- **5. break Statement**
  - **Stops** the loop immediately.
- **Example:**

```
for i in range(1, 10):
    if i == 5:
        break
    print(i)
```

- **6. continue Statement**
  - **Skips** the current iteration and moves to the next.
- **Example:**

```
for i in range(1, 6):
    if i == 3:
        continue
    print(i)
```

🔹 7. Nested Loops

A loop **inside another loop**.

🔸 Example:

```
for i in range(1, 4):
    for j in range(1, 3):
        print(i, j)
```

🔸 Suggested Programs

☑ Program 1: Print a Pattern

```
*
* *
* * *
* * * *

for i in range(1, 5):
    print("* " * i)
```

☑ Program 2: Summation of Series

```
# Sum of first n natural numbers
n = int(input("Enter a number: "))
sum = 0
for i in range(1, n+1):
    sum += i
print("Sum =", sum)
```

☑ Program 3: Factorial of a Number

```
n = int(input("Enter a number: "))
fact = 1
for i in range(1, n+1):
    fact *= i
print("Factorial =", fact)
```

☑ Program 4: Print even numbers between 1 and 20

```
for i in range(2, 21, 2):
    print(i)
```

☑ **Summary Table**

Loop Type	Use When
for	Number of repetitions is known (fixed count)
while	Repeat until condition is false (unknown count)
break	Exit the loop immediately
continue	Skip current iteration
nested	Loop inside loop (e.g., for patterns)

Practice Q.

1. Write a program to print numbers from **1 to 10** using a `for` loop.
2. Print the **first 10 even numbers** using a `for` loop.
3. Print the **table of 5** using a `for` loop.

4. Use the `range()` function to print numbers from **10 to 1** in reverse.
5. Print all numbers between **50 and 100** which are divisible by **7**.

6. Write a program to print numbers from **1 to 5** using a `while` loop.
7. Take a number from the user and print whether it is **prime or not**.

8. Use a loop to print numbers from **1 to 10**, but **stop the loop if number is 6**.
9. Print numbers from **1 to 10**, but **skip number 5** using `continue`.

10. Write a program to print the following pattern:

```
*
* *
* * *
* * * *
```

11. Print the following pattern:

```
1
1 2
1 2 3
1 2 3 4
```

12. Write a program to calculate the **sum of first N natural numbers** (where N is user input).
13. Write a program to find the **factorial of a number** entered by the user.
14. Write a program to calculate the **sum of squares** of first 10 natural numbers.